

Improved Solution to the Non-Domination Level Update Problem

Sumit Mishra

Department of Computer Science
& Engineering
Indian Institute of Technology Patna
Patna, Bihar India - 800013
sumitmishra@iitp.ac.in

Samrat Mondal

Department of Computer Science
& Engineering
Indian Institute of Technology Patna
Patna, Bihar India - 800013
samrat@iitp.ac.in

Sriparna Saha

Department of Computer Science
& Engineering
Indian Institute of Technology Patna
Patna, Bihar India - 800013
sriparna@iitp.ac.in

Abstract—Non-domination level update problem is to sort the non-dominated fronts after insertion or deletion of a solution. Generally the solution to this problem requires to perform the complete non-dominated sorting which is too expensive in terms of number of comparisons. Recently an Efficient Non-domination Level Update (ENLU) approach is proposed which does not perform the complete sorting. For this purpose, in this paper a space efficient version of ENLU approach is proposed without compromising the number of comparisons. However this approach does not work satisfactorily in all the cases. So we have also proposed another tree based approach for solving this non-domination level update problem. In case of insertion, the tree based approach always checks for same number of fronts unlike linear approach in which the number of fronts to be checked depends on the inserted solution. The result shows that in case where all the solutions are dominating in nature the maximum number of comparisons using tree based approach is $\mathcal{O}(\log N)$ as opposed to $\mathcal{O}(N)$ in ENLU approach. When all the solutions are equally divided into K fronts such that each solution in a front is dominated by all the solutions in the previous front then the maximum number of comparisons to find a deleted solution in case of tree based approach is $K - \log K$ less than that of ENLU approach. Using these approaches an on-line sorting algorithm is also proposed and the competitive analysis of this algorithm is also presented.

I. INTRODUCTION

In past few decades the evolutionary algorithms [1], [2], [3], [4] have gained a lot of popularity. One of the primary reason behind its popularity is their ability to solve real world problems. Real world problems may involve simultaneous optimizing multiple objectives. Thus the evolutionary algorithms also optimize single as well as multiple objectives. In case of single objective optimization, only single solution is optimal one. But in case of multi-objective optimization problems (MOOPs) [5], [6], [7], [8] a set of set of optimal solutions are achieved and these are known as Pareto-optimal solutions.

In literature various multi-objective evolutionary algorithms (MOEAs) are proposed. Some of them are non-dominated sorting genetic algorithm II (NSGA-II) [9], strength pareto evolutionary algorithm 2 (SPEA2) [10], pareto archive evolution strategy (PAES) [11], pareto envelope-based selection algorithm (PESA) [12] and pareto frontier differential evolution (PDE) [13] etc. These MOEAs are able to find a set of Pareto optimal solutions in one single run. There exist various approaches for

Front	Solutions
F_1	$sol_1, sol_2, \dots, sol_{n_1}$
F_2	$sol_{n_1+1}, sol_{n_1+2}, \dots, sol_{n_1+n_2}$
\vdots	\vdots
F_K	$sol_{n_1+\dots+n_{K-1}+1}, \dots, sol_N$

TABLE I: Solutions in K different fronts

the selection of solutions [14]. But the Pareto-based approach is generally used. Non-dominated sorting [9] is found to be efficient for finding Pareto-optimal solution out of various techniques.

In this sorting the solutions are assigned to their respective front based on their dominance relationship. This process is time consuming when the number of solutions in the populations becomes larger. Much work [9], [15], [16] has been done to improve the running time of this process. Goldberg et al. [17] first proposed the idea of non-dominated sorting. Later this idea was used in multi-objective genetic algorithm [18].

Let $\mathbb{P} = \{p_1, p_2, \dots, p_N\}$ be the population of N solutions. These solutions are categorized into K fronts. These K fronts are denoted as $F_k, 1 \leq k \leq K$. The solutions which belong to front F_k are dominated by at-least one of the solutions belonging to front $F_{k'}, k' < k, k, k' = 1, 2, \dots, K$. Let the number of solutions in each front F_k is $n_k, 1 < k < K$. Thus $N = \sum_{k=1}^K n_k$. The arrangement of the solutions in each front is shown in Table I. Consider an example.

Example 1.1: Let \mathbb{P} be a set of 12 solutions. Two objectives are associated with each solution. Let both the objectives are to be minimized. These 12 solutions are arranged in 5 fronts. The arrangement of solutions in each fronts is as follows: $F_1 = \{p_1\}$, $F_2 = \{p_2, p_3\}$, $F_3 = \{p_4, p_5, p_6, p_7\}$, $F_4 = \{p_8, p_9, p_{10}, p_{11}\}$, $F_5 = \{p_{12}\}$.

The generational Evolutionary Multi-Objective Optimization (EMO) algorithms generate all the offspring solutions from the parent solutions. Then they both are compared. As opposite to the generational EMO algorithms, steady state EMO algorithms [19], [20] update the parent population as a new offspring is derived. The steady state EMO algorithms have the ability to generate the good offspring solutions. The parallel implemen-

tation is also possible with this kind of EMO algorithms.

There have not been so many such kind of EMO algorithms proposed [21]. One of the primarily reason for this is the overhead in repeatedly performing the non-dominated sorting as a new solution is generated or an existing solution is removed. But when either a new solution is inserted or an existing solution is removed then not all the solutions change its domination level so it is unnecessary to perform the complete non-dominated sorting again and again. Only some of the solutions need to change their non-domination level. This is first addressed in [21]. Li et al. [21] proposed an efficient non-domination level update (ENLU) approach for steady-state EMO algorithms. They have proposed the approach for insertion as well as deletion. In this approach not all the solutions change their domination level. The solutions which need to change their domination level, only change the level. After this some more work has been carried out in this direction [22], [23]. But these two work focus on bi-objective steady state EMO algorithms. So in this paper we have proposed the modified version of ENLU which is efficient in terms of space and time complexity remains the same. One more approach is provided which is based on tree data structure. The maximum number of dominance comparison while performing insertion or deletion is also obtained. In short the main contribution are as follows:

- The modified linear approach with space requirement $\mathcal{O}(1)$ as opposite to $\mathcal{O}(N)$ in [21] is proposed.
- The dominance tree based approach is proposed for non-domination level update problem.
- We have obtained the maximum number of dominance comparisons occurred in linear as well as dominance tree based approach.
- The approach for searching a solution is also proposed using dominance tree.
- The solution to the non-domination level update problem can be used as a non-dominated sorting algorithm. This sorting algorithm can be used as on-line algorithm. The competitive ratio of this on-line algorithm is also obtained.

The rest of this paper is organized as follows. Section II describes the Non-domination Level Update problem. The related work is also described in this Section. The modified linear approach to insert a solution in the given set of fronts is discussed in Section III. The Proposed dominance binary search tree based approach is illustrated in Section IV. The look up procedure using dominance binary search tree based approach is provided in Section V. The procedure to delete an existing solution is discussed in Section VI. Section VII discuss the maximum number of dominance comparison in Non-domination Level Update problem. In section VIII and IX we discuss the maximum and minimum number of dominance comparison in two cases when all the solutions are non-dominating and when all are dominating respectively. The sorting algorithm using the proposed approach is presented in Section X. Finally Section XI concludes the paper and provides the future direction of the work.

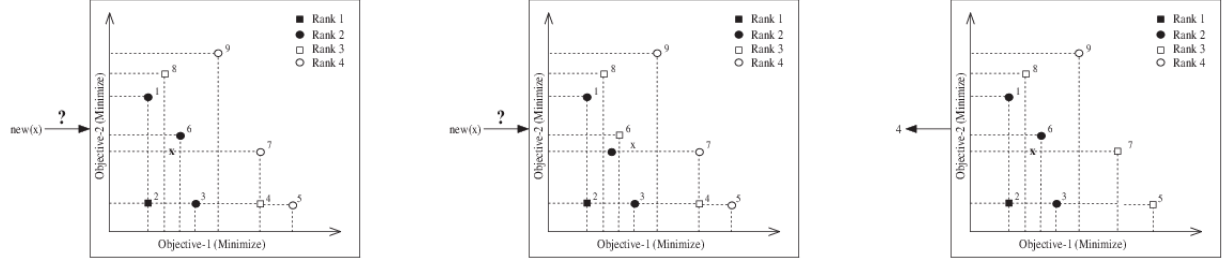
II. NON-DOMINATION LEVEL UPDATE PROBLEM

In this section we discuss the non-domination level update problem. Non-domination Level Update problem is to sort the non-dominating front after insertion of a new solution or after deletion of an existing solution. Let $\mathbb{P} = \{sol_1, sol_2, \dots, sol_N\}$ be the set of N solutions. Let M objectives are associated with these N solutions. These N solutions are divided into K fronts. Let $\mathcal{F} = \{F_1, F_2, \dots, F_{K-1}, F_K\}$ be the set of K fronts in decreasing order of their dominance i.e., the first front is having rank 1 (non-domination level 1), second is having rank 2 and so on. The number of solutions in each front F_i is given by n_i , $1 \leq i \leq K$. Thus $\sum_{i=1}^K n_i = N$. Table I shows this scenario.

Let a new solution *new* is to be inserted into the set of K fronts. Non-domination Level Update problem in case of insertion is to insert this new solution at its correct position in the set of fronts and update the fronts if needed. Figure 1a clearly shows this situation. In this figure, there are 9 solutions. Let $\mathbb{P} = \{1, 2, \dots, 9\}$ be the set of these 9 solutions. These solutions are divided into 4 fronts. Let $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$ be the set of 4 fronts in the decreasing order of their dominance. $F_1 = \{2\}$, $F_2 = \{1, 3, 6\}$, $F_3 = \{4, 8\}$, $F_4 = \{5, 7, 9\}$. A new solution *new* is to be inserted in the set of fronts. Main aim here is to update \mathcal{F} after insertion of *new*. Figure 1b shows the updated set of fronts when new solution is inserted in the set of fronts.

Let an existing solution *sol* is to be deleted from the set of K fronts. Non-domination Level Update problem in case of deletion is to delete this solution and update the set of fronts if required. Figure 1a shows the set of 4 fronts. An existing solution 4 is to be deleted from the set of fronts. The motive is to update \mathcal{F} after deletion of solution 4. Figure 1c shows the updated set of fronts when solution 4 is being deleted from the set of fronts.

The naive approach is to apply the non-dominated sorting algorithm on all the N solutions along with new solution in case of insertion or apply the non-dominated sorting algorithm on the remaining $N-1$ solutions in case of deletion. Thus the complete sorting algorithm is to be applied on either $N+1$ or $N-1$ solutions. If we use the brute-force technique then the time complexity will be $\mathcal{O}(MN^3)$ and space requirement will be $\mathcal{O}(N)$. When the fast non-dominating sorting proposed in [9] is used then the time complexity will be $\mathcal{O}(MN^2)$. This always requires $(N+1)N$ dominance comparison in case of insertion and $(N-1)(N-2)$ comparison in case of deletion regardless of arrangement of fronts. The space requirement will be $\mathcal{O}(N^2)$. Tang et al. [24] proposed arena's principle. The time complexity of their approach is $\mathcal{O}(MN^2)$ in worst case. The space requirement is $\mathcal{O}(N)$. McClymont et al. [15] proposed two sorting algorithms - climbing sort and deductive sort. The climbing sort requires $(N+1)N$ comparison in case of insertion and $(N-1)(N-2)$ comparison in case of deletion when the worst case scenario occurs. The space requirement is $\mathcal{O}(N^2)$. The deductive sort has some advantage over climbing. In the worst case the number of comparison is $\frac{(N+1)N}{2}$ for



(a) Set of fronts \mathcal{F} before insertion of *new*. (b) Set of fronts \mathcal{F} after insertion of *new*. (c) Set of fronts \mathcal{F} after deletion of 4.

Fig. 1: Schematic diagram of Non-domination Level Update problem in case of insertion.

insertion and $\frac{(N-1)(N-2)}{2}$ for deletion. The space requirement is $\mathcal{O}(N)$ as opposite to $\mathcal{O}(N^2)$ by climbing sort. The approach proposed in [16] requires $\mathcal{O}(MN^2)$ time in worst case. In the worst case the total number of dominance comparison is same as deductive sort. This approach first sorts the solutions based on first objective which also requires $\mathcal{O}(N \log N)$ time. The space requirement of this approach is $\mathcal{O}(1)$.

All the above discussed approaches apply the complete sorting algorithm on either $N+1$ or $N-1$ solutions. To reduce the number of dominance comparisons without applying the complete sorting algorithm, Li et al. [21] proposed an approach for Non-domination Level Update problem. The approach was proposed for Steady-State Evolutionary Multiobjective Optimization. The time complexity is $\mathcal{O}(MN\sqrt{N})$ in case of equal division of solutions in \sqrt{N} fronts. However the worst case time complexity is $\mathcal{O}(MN^2)$ [22], [23]. The space requirement is $\mathcal{O}(N)$ as the dominated solution is kept at two places - one is in archive S and other is in front F_{i+1} .

Yakupov et al. [22] proposed an incremental non-dominated sorting for bi-objective solutions. The running time of the approach is $\mathcal{O}(M(1 + \log(N/M)) + \log M \log(N/\log M))$ which is $\mathcal{O}(N)$ in worst case when a new solution is inserted in the set of fronts having N solutions with M objectives. With further improvement in this work, Buzdalov et al. [23] proposed fast implementation of the steady-State NSGA-II algorithm for two dimensions based on incremental non-dominated sorting. In this work, the support of the crowding distance calculation is employed. After that, the steady-state version of the NSGA-II algorithm is presented. In [22] and [23] a data structure tree of tress is used. The nodes in higher level tree stores the number of tree elements in a sub-tree, the previous-in-order node and the next-in-order node. Thus extra space is required which is $\mathcal{O}(K)$. The nodes in lower level tree also stores the number of tree elements in a sub-tree, the previous-in-order node and the next-in-order node. Thus extra space is required which is $\mathcal{O}(N)$. Thus the space requirement is $\mathcal{O}(K) + \mathcal{O}(N) \equiv \mathcal{O}(N)$.

We have proposed an approach based on dominance binary search tree. First we discuss the modified version of linear approach proposed in [21] for insertion as well as deletion. The modification is done to reduce the space complexity from $\mathcal{O}(N)$ to $\mathcal{O}(1)$.

III. MODIFIED LINEAR APPROACH: INSERT A SOLUTION

In this section we will discuss the Non-domination Level Update problem using linear way to obtain the correct position of new solution *new* as done in [21]. Algorithm 1 shows the insertion procedure of *new* in the list of non-dominated fronts \mathcal{F} . This algorithm does not run the complete sorting algorithm again. It uses the non-dominance properties of the solution in the same front and uses the ranking of the front. Here we are assuming that all the fronts are arranged in decreasing order of their dominance. The process for inserting a solution is summarized in Algorithm 1. Two solutions are compared for dominance nature using *domNature(A,B)* procedure which returns the following three values.

- 1: Solution *A* dominates *B*.
- -1: Solution *A* is dominated by *B*.
- 0: Solution *A* and *B* are non-dominated.

The new solution *new* is compared with all the fronts in sequential manner starting from \mathcal{F}_1 to \mathcal{F}_K . *new* is compared with each solution in a front. When *new* is compared with any solution in a front $F_k (1 \leq k \leq K)$ then there are three possibilities:

1. If the solution in the front F_k dominates *new* it means *new* can not be inserted into F_k . So now we check for another front having lower dominance (F_{k+1}) without checking it with other solutions in the same front. If $i = K$ then *new* creates a new front F_{K+1} having lowest dominance among all the fronts.
2. If the solution in the front is non-dominating with the *new* then we keep on comparing *new* with rest of the solutions in that front. If *new* is non-dominating with all the solutions in the front F_k then *new* is added in the front F_k .
3. If *new* dominates the solution in the front F_k then we obtain the list *new_{dom}* of all the solutions in the front F_k which are dominated by *new* using *DomSet()* procedure as described in Algorithm 2. This *DomSet()* procedure returns the list of all the solutions in front F_k which are dominated by *new*. The solutions which are dominated by *new* are removed from front F_k . After the removal of dominated solution in F_k , *new* is added to F_k . Now we have the following possibilities:

- If $k = K$ then *new_{dom}* creates a new front F_{K+1} having lowest dominance among all the fronts.

Algorithm 1 InsertLinear(\mathcal{F} , new)

Input: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$: Non-dominated fronts in the decreasing order of their dominance

new : A new solution

Output: Updated \mathcal{F} after insertion of new

```
1: for  $i \leftarrow 1$  to  $K$  do
2:    $count \leftarrow 0$ 
3:    $new_{dom} \leftarrow \Phi$ 
4:   for  $j \leftarrow 1$  to  $|F_i|$  do
5:      $isdom \leftarrow domNature(new, F_i(j))$ 
6:     if  $isdom = 1$  then
7:        $new_{dom} \leftarrow new_{dom} \cup \{F_i(j)\}$ 
8:        $F_i \leftarrow F_i \setminus \{F_i(j)\}$ 
9:        $DomSet(F_i, new, i, new_{dom})$ 
10:       $F_i \leftarrow F_i \cup new$ 
11:      if  $i = K$  then
12:        Make  $new_{dom}$  a new front  $F_{K+1}$ 
13:      else if  $|F_i| = 1$  then
14:        Increase the dominance level of all the fronts
         $F_{k+1}, F_{k+2}, \dots, F_K$  by 1 and make  $new_{dom}$  a
        new front  $F_{k+1}$ 
15:      else
16:         $UpdateInsert(\mathcal{F}, new_{dom}, i+1)$ 
17:      return  $\mathcal{F}$ 
18:    else if  $isdom = 0$  then
19:       $count \leftarrow count + 1$ 
20:    else
21:      break
22:  if  $count = |F_i|$  then
23:    Insert  $new$  in  $F_i$  /*  $new$  is non-dominated with  $F_i$ 
24:  return  $\mathcal{F}$ 
25: Make  $new$  a new front  $F_{K+1}$ 
26: return  $\mathcal{F}$ 
```

Algorithm 2 DomSet(F , new , $index$, S)

Input: F : A non-dominated front

new : A new solution

$index$: Index of the solution from where the dominance need to be checked

S : Set of solutions dominated by new

Output: Updated S

```
1: for  $i \leftarrow index$  to  $|F|$  do
2:    $isdom \leftarrow domNature(new, F(i))$ 
3:   if  $isdom = 1$  then
4:      $S \leftarrow S \cup \{F(i)\}$ 
5:      $F \leftarrow F \setminus F(i)$ 
6:      $i \leftarrow i - 1$ 
7: return  $S$ 
```

- If new dominates all the solutions in the front (after removing the dominated solutions from F_k and adding new to F_k the cardinality of F_k is 1) i.e. $|F_k| = 1$ then the dominance level of all the fronts $F_{k+1}, F_{k+2}, \dots, F_K$ are increased by one and dominated solution new_{dom} is assigned to front F_{k+1} .
- If none of the above two conditions are satisfied i.e. new dominates some of the solutions in front F_k and $k < K$ then $UpdateInsert()$ procedure is used which re-arranges the solutions in their respective front. This $UpdateInsert()$ procedure is discussed in Algorithm 3.

A. Illustration of DomSet(F , new , $index$, S) procedure

This procedure takes as input a front (F), new solution (new), the index of the solution ($index$) in the front from where the dominance of new needs to be checked and set of solutions dominated by new (S). This procedure returns the updated set of solutions dominated by new in front F . For this purpose new is checked against all the solutions in the front starting from index position. The solution which is dominated by new is added to S and removed from the front F . We are removing the dominated solutions from the front to save the space. In this manner same solution does not occupy more than one place.

Complexity Analysis: In the worst case, all the solutions in the front except first is compared with new solution in the DomSet() procedure so the worst case complexity of DomSet() procedure becomes $\mathcal{O}(M|F|)$.

Algorithm 3 UpdateInsert(\mathcal{F} , S , $index$)

Input: \mathcal{F} : Set of non-dominated fronts

$index$: Non-domination level index

Output: Updated \mathcal{F}

```
1:  $l \leftarrow |S|$ 
2: for  $i \leftarrow 1$  to  $|F_{index}|$  do
3:    $count \leftarrow 0$ 
4:   for  $j \leftarrow 1$  to  $l$  do
5:     if  $domNature(S(j), F_{index}(i)) = 0$  then
6:        $count \leftarrow count + 1$ 
7:   if  $count = l$  then
8:      $S \leftarrow S \cup F_{index}(i)$ 
9:      $F_{index} \leftarrow F_{index} \setminus F_{index}(i)$ 
10:     $i \leftarrow i - 1$ 
11: if  $l = |S|$  then
12:   Increase the domination level of  $F_k, k \in \{index, index+1, \dots, K\}$  by 1 and make  $S$  a new front
    $F_{index}$ 
13: else if  $F_{index} = \Phi$  then
14:   Make  $S$  as  $F_{index}$ 
15: else
16:    $T \leftarrow F_{index}$  /*Move the solutions from  $F_{index}$  to  $T$ 
17:   Make  $S$  as  $F_{index}$ 
18:    $UpdateInsert(\mathcal{F}, T, index+1)$ 
```

B. Illustration of UpdateInsert($\mathcal{F}, S, index$) procedure

This procedure takes as input the set of non-dominated fronts \mathcal{F} , a set of solution S , the index of the front denoted by $index$ in \mathcal{F} . This procedure updates the \mathcal{F} by either creating a new front or by re-arranging the solutions in the existing fronts. First of all the initial cardinality of S is stored is l . This is because when the solutions from F_{index} is compared with S for non-dominance then it should be compared with only first l solutions.

Here we find the solutions in F_{index} which are non-dominated with S . The solutions which are non-dominated with S are added to it and removed from F_{index} . The removal guarantees that no solution can occupy more than one place. Now the following situation can occur:

- If no solution from F_{index} is being added to S i.e. $l = |S|$ then the dominance level of fronts $F_k, k \in \{index, index+1, \dots, K\}$ is increased by 1 and S is assigned the dominance level $index$.
- If all the solutions from F_{index} is added to S i.e. $F_{index} = \Phi$ then make S as F_{index} .
- Otherwise all the solutions from F_{index} is moved to T . The movement means as a solution from F_{index} is moved to T , the solution is being removed from F_{index} . The non-domination level of F_{index} is assigned to S . The procedure is repeated with UpdateInsert($\mathcal{F}, T, index+1$).

Complexity Analysis: In this algorithm the maximum number of comparison is performed when *new* dominates the solutions in the first front F_1 i.e. this procedure is called with $index = 2$. For maximum number of comparison, the $|new_{dom}| = n_1 - 1$. The maximum number of comparison occurs when each call to this procedure shifts one solution in higher level front after comparing with all the solutions. Thus the maximum number of comparison for this procedure is given by $(n_1 - 1)n_2 + (n_2 - 1)n_3 + \dots + (n_{k-1} - 1)n_K$. Each comparison between two solutions requires at-most M comparison between M objectives. Thus the worst case complexity of this procedure is $\mathcal{O}(MN^2)$. The best case occurs when F_1 has single solution and *new* is non-dominating with this solution. In this case only one comparison is required so the best case complexity is $\mathcal{O}(M)$.

C. Complexity of Modified Linear Approach

Here we will analyze the complexity using the linear approach. We can see from all the algorithms 1, 2 and 3 that no solution is being kept at more than one place. Only few scalar variables are required. Therefore, the space complexity of linear approach is $\mathcal{O}(1)$. The worst case time complexity of UpdateInsert() procedure dominates the worst case time complexity of DomSet() procedure. The complexity of UpdateInsert() procedure is quadratic while DomSet() has linear complexity. Thus the overall worst case complexity of the modified approach is $\mathcal{O}(MN^2)$. This time complexity is same as proposed in [21].

IV. PROPOSED APPROACH USING DOMINANCE BINARY SEARCH TREE

In this section we will discuss the proposed approach for Non-domination Level Update problem using dominance tree. This approach inserts the new solution to its correct position using tree based methodology unlike linear way as proposed in [21]. The only difference between the linear approach and tree based approach is in identifying the position of the inserted solution. The update procedure remains the same. Thus dominance tree based approach will be beneficial when there are large number of fronts. We first provide formal definition of this tree. Two variants of this tree are also discussed.

The tree based data structure is also used in [22] and [23]. The authors have used tree of trees where the high level tree corresponds to the non-dominated front i.e. node in the high level tree corresponds to a front. The low level tree corresponds to the solutions inside a front i.e. the nodes in the low level tree corresponds to solutions inside the front. The solutions in the low level tree are sorted according to first objective. The high-level tree can be an ordinary balanced tree while the low level tree should be a split-merge tree. The uses of this data structure perform some of the operations in $\mathcal{O}(\log N)$ time like element search in the container, splitting of container by key into two parts and merging of two container. In [23], Cartesian Tree [25] is used as a split-merge tree as it performs better than the Splay Tree [26] in practice.

We have used only high level tree named as Dominance Binary Search Tree. The solutions inside a node are arranged in linear fashion. Thus no sorting is required.

A. Dominance Binary Search Tree

In *Dominance Binary Search Tree*, the node represents single non-dominated front i.e., the solutions in a node are non-dominating with each other.

Definition 1 (Dominance Binary Search Tree): A binary tree T is known as *Dominance Binary Search Tree* if the node is having lower dominance than left sub-tree and higher dominance than right sub-tree.

Example 4.1: Let $\mathcal{F} = \{F_1, F_2, \dots, F_{15}\}$ be the set of 15 non-dominated fronts in decreasing order of their dominance i.e., the first front is having rank 1 (higher rank), second is having rank 2 and so on. The corresponding Dominance Binary Search Tree is given in Figure 2a.

Dominance binary search tree can be categorized into two types based on how the root of a sub-tree is chosen.

- 1) **Left-Balanced Dominance Binary Search Tree:** If left child of each node is filled before the right child then the tree is called left-balanced dominance binary search tree. It is obtained when the index of the root of a sub-tree is calculated as $mid = \lceil (min + max)/2 \rceil$. See Figure 2b.
- 2) **Right-Balanced Dominance Binary Search Tree:** If right child of each node is filled before the left child then the tree is called right-balanced dominance binary search tree. It is obtained when the index of the root of a sub-tree is calculated as $mid = \lfloor (min + max)/2 \rfloor$. See Figure 2c.

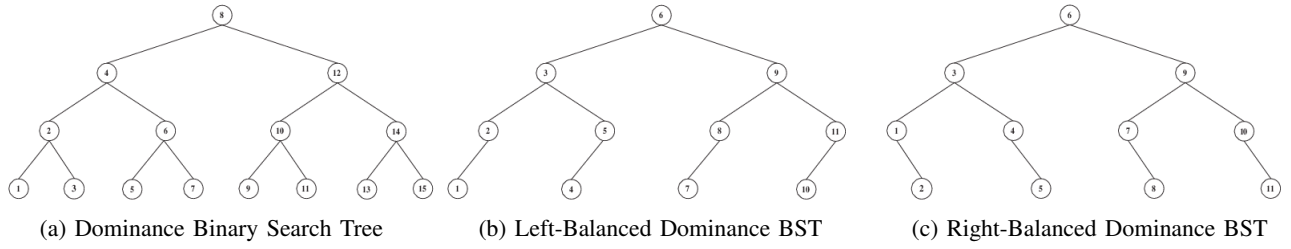


Fig. 2: Dominance Binary Search Tree

Unlike the linear approach where *new* needs to be compared with all the K fronts in the worst case, here the solution is compared with either $\lfloor \log_2 K \rfloor + 1$ or $\lfloor \log_2 K \rfloor$ fronts in all the cases. The process to insert a new solution *new* in the set of fronts \mathcal{F} is summarized in Algorithm 4.

B. Illustration of Algorithm 4

Algorithm 4 shows the insertion procedure of a new solution in the fronts using dominance binary search tree approach. Here like linear approach, we are also assuming that all the fronts are arranged in decreasing order of their dominance. This insertion can make the following changes in the tree.

- I. *new* can make new front i.e., creation of a new node.
- II. *new* can be merged with any of the existing fronts.
- III. *new* can be merged with any of the fronts by removing some of the solutions (which are dominated by new_{dom}) in that front. After this removal, the solutions in the lower dominance fronts are re-arranged.

With the help of Algorithm 4 a new solution *new* is inserted in the given list of fronts. If there is only single front then Algorithm 1 is called. Here the insertion is performed in the same manner as in linear approach. So in case of single front i.e. when all the solutions are non-dominating in nature then the linear approach and dominance tree based approach is same. If the number of fronts are more than one then the dominance binary search tree approach comes into picture. This algorithm first executes Algorithm 5 which gives a list known as *CmpFront*. In place of Algorithm 5 which is based on Left Dominance Binary Search Tree, Algorithm 6 can also be used which is based on Right Dominance Binary Search Tree.

Algorithm 5 and 6 returns a list *CmpFront*. Each element of this list is 3-folded - $\langle \text{dom}, fIndex, sIndex \rangle$. ‘dom’ is the dominating nature of *new* with a particular solution in the front. ‘fIndex’ is the index of the front to which the *new* is compared and ‘sIndex’ is the index of the solution in F_{fIndex} to which *new* is compared and ‘dom’ is achieved. Thus ‘dom’ shows the dominated nature of *new* with $F_{fIndex}(sIndex)$. The ‘dom’ can take three values $-1, 0, 1$. The value of ‘fIndex’ can vary between 1 to K . Generally the value of ‘sIndex’ can vary between 1 to n_k , $1 \leq k \leq K$ but when *new* is non-dominating with all the solutions in a particular front then the value of ‘sIndex’ is 0. The maximum length of this list is $\lfloor \log_2 K \rfloor + 1$ because the maximum number of fronts to which *new* can be compared is $\lfloor \log_2 K \rfloor + 1$.

Algorithm 4 InsertTree(\mathcal{F}, new)

Input: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$: Non-dominated front in the decreasing order of their dominance

new: A new solution

Output: Updated \mathcal{F} after insertion of *new*

```

1: if  $|\mathcal{F}| = 1$  then
2:   Insert( $\mathcal{F}, new$ )           /* Same as linear approach
3: else
4:    $CmpFront[] \leftarrow \langle \text{dom}, fIndex, sIndex \rangle \leftarrow \Phi$ 
5:   InsertTree_Left( $\mathcal{F}, 1, |\mathcal{F}|, new$ )
6:    $len \leftarrow CmpFront.size()$ 
7:   if  $CmpFront[len]_{\text{dom}} = 0$  then
8:     Insert new in  $F_{CmpFront[len]_{fIndex}}$ 
9:   else if  $CmpFront[len]_{\text{dom}} = 1$  then
10:     $new_{dom} \leftarrow \Phi$ 
11:     $fIndex \leftarrow CmpFront[len]_{fIndex}$ 
12:     $sIndex \leftarrow CmpFront[len]_{sIndex}$ 
13:     $new_{dom} \leftarrow new_{dom} \cup F_{fIndex}(sIndex)$ 
14:     $F_{fIndex} \leftarrow F_{fIndex} \setminus F_{fIndex}(sIndex)$ 
15:    DomSet( $F_{fIndex}, new, sIndex+1, new_{dom}$ )
16:     $F_{fIndex} \leftarrow F_{fIndex} \cup new$ 
17:    if  $fIndex = K$  then
18:      Make  $new_{dom}$  a new front  $F_{fIndex+1}$ 
19:    else if  $|F_{fIndex}| = 1$  then
20:      Increase the dominance level of all fronts
         $F_{fIndex+1}, F_{fIndex+2}, \dots, K$  by 1 and make  $new_{dom}$ 
        a new front  $F_{fIndex+1}$ 
21:    else
22:      Update( $\mathcal{F}, new_{dom}, fIndex+1$ )
23:  else if All the dom value in CmpFront is -1 then
24:    Make  $new_{dom}$  a new front  $F_{K+1}$ 
25:  else
26:    for  $i \leftarrow len$  to 2 do
27:      if  $CmpFront[i]_{\text{dom}} \neq CmpFront[i-1]_{\text{dom}}$  then
28:         $fIndex \leftarrow CmpFront[i-1]_{fIndex}$ 
29:         $sIndex \leftarrow CmpFront[i-1]_{sIndex}$ 
30:        if  $CmpFront[i-1]_{\text{dom}} = 0$  then
31:          Insert new in  $F_{fIndex}$ 
32:        else if  $CmpFront[i-1]_{\text{dom}} = 1$  then
33:           $new_{dom} \leftarrow \Phi$ 
34:           $new_{dom} \leftarrow new_{dom} \cup F_{fIndex}(sIndex)$ 
35:           $F_{fIndex} \leftarrow F_{fIndex} \setminus F_{fIndex}(sIndex)$ 
36:          DomSet( $F_{fIndex}, new, sIndex+1, new_{dom}$ )
37:          UpdateInsert( $\mathcal{F}, new_{dom}, fIndex$ )

```

Algorithm 5 InsertTree_Left(\mathcal{F} , min , max , new)

Input: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$: Non-dominated front in the decreasing order of their dominance new : A new solution**Output:** $CmpFront$

```
1:  $count \leftarrow 0$ 
2: if  $min = max$  then
3:   for  $i \leftarrow 1$  to  $|F_{min}|$  do
4:      $isdom \leftarrow \text{domNature}(new, F_{min}(i))$  /* check the dominating nature of  $new$  and  $F_{min}(i)$  */
5:     if  $isdom = 1$  then /*  $new$  dominates  $F_{min}(i)$  */
6:        $CmpFront.add(1, min, i)$  /* Add dominating nature of  $new$ , front index and solution index */
7:       break
8:     else if  $isdom = -1$  then /*  $new$  is dominated by  $F_{min}(i)$  */
9:        $CmpFront.add(-1, min, i)$  /* Add dominating nature of  $new$ , front index and solution index */
10:      break
11:    else /*  $new$  and  $F_{min}(i)$  are non-dominating */
12:       $count \leftarrow count + 1$ 
13:    if  $count = |F_{min}|$  then
14:       $CmpFront.add(0, min, 0)$  /* Add dominating nature of  $new$ , front index and solution index */
15:  else
16:     $mid \leftarrow \lceil (min + max) / 2 \rceil$  /* Obtain the position of the node to be explored */
17:    for  $i \leftarrow$  to  $|F_{mid}|$  do
18:       $isdom \leftarrow \text{domNature}(new, F_{mid}(i))$  /* check the dominating nature of  $new$  and  $sol$  */
19:      if  $isdom = 1$  then /*  $new$  dominates  $sol$  */
20:         $CmpFront.add(isdom, mid, i)$  /* Add dominating nature of  $new$ , front index and solution index */
21:        InsertTree_Left( $\mathcal{F}$ ,  $min$ ,  $mid-1$ ,  $new$ ) /* Explore left sub-tree */
22:        break
23:      else if  $isdom = -1$  then /*  $F_{mid}(i)$  dominates  $new$  */
24:         $CmpFront.add(isdom, mid, i)$  /* Add dominating nature of  $new$ , front index and solution index */
25:        if  $mid \neq max$  then /* Check for the existence of right sub-tree */
26:          InsertTree_Left( $\mathcal{F}$ ,  $mid+1$ ,  $max$ ,  $new$ ) /* Explore right sub-tree */
27:          break
28:      else /*  $new$  and  $F_{mid}(i)$  are non-dominating */
29:         $count \leftarrow count + 1$ 
30:      if  $count = |F_{mid}|$  then /* All solutions in  $F_{mid}$  are non-dominating with  $new$  */
31:         $CmpFront.add(0, mid, 0)$  /* Add dominating nature of  $new$ , front index and solution index */
32:        InsertTree_Left( $\mathcal{F}$ ,  $min$ ,  $mid-1$ ,  $new$ ) /* Explore left sub-tree */
```

After obtaining $CmpFront$, the correct position of new is to be identified. Let the length of $CmpFront$ is denoted by len . There are following possibilities:

- 1) If last node in the $CmpFront$ is non-dominating with new then new will be inserted in the front corresponding to last node. The index of the front corresponding to last node is obtained by $CmpFront[len]_{fIndex}$.
- 2) If new dominates last node in the $CmpFront$ then
 - I. Obtain the index of the front corresponding to last node. The index of this node, $fIndex = CmpFront[len]_{fIndex}$.
 - II. Obtain the index of the solution which is first dominated by new in the F_{fIndex} . The index of this solution, $sIndex = CmpFront[len]_{sIndex}$.
 - III. Add the solution $F_{fIndex}(sIndex)$ in new_{dom}
 - IV. Remove the solution $F_{fIndex}(sIndex)$ from F_{fIndex}
 - V. Obtain rest of the dominated solution in F_{fIndex} by new using $DomSet()$ procedure. The $DomSet()$ is called with

following parameter - F_{fIndex} , new , $sIndex+1$ and new_{dom} .

- VI. After obtaining new_{dom} , new is being added to F_{fIndex} . Now one of the following three possibilities may arise:

- If $fIndex = K$ then new_{dom} creates a new front F_{K+1} having lowest dominance.
- If new dominates all the solutions in front F_{fIndex} i.e. $|F_{fIndex}| = 1$ then the dominance level of all the fronts $F_{fIndex+1}, F_{fIndex+2}, \dots, F_K$ are increased by 1 and new_{dom} is assigned to front $F_{fIndex+1}$.
- If none of the above two conditions are satisfied then $UpdateInsert()$ procedure is used which rearranges the solutions in their respective fronts. The $UpdateInsert()$ is called with following parameter - \mathcal{F} , new_{dom} and $fIndex+1$.

- 3) If the values denoting dominance nature of new in $CmpFront$ list contains -1 for all the fronts i.e. new is dominated by all the compared fronts. Formally $CmpFront[i]_{dom} = -1 \forall i, 1 \leq$

Algorithm 6 InsertTree_Right(\mathcal{F} , min , max , new)

Input: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$: Non-dominated front in the decreasing order of their dominance new : A new solution**Output:** $CmpFront$

```
1:  $count \leftarrow 0$ 
2: if  $min = max$  then
3:   for  $i \leftarrow$  to  $|F_{min}|$  do
4:      $isdom \leftarrow \text{dominates}(new, F_{min}(i))$  /* check the dominating nature of  $new$  and  $F_{min}(i)$  */
5:     if  $F_{min}(i) = 1$  then /*  $new$  dominates  $sol$  */
6:        $CmpFront.add(1, min, i)$  /* Add dominating nature of  $new$ , front index and solution index */
7:       break
8:     else if  $isdom = -1$  then /*  $new$  is dominated by  $F_{min}(i)$  */
9:        $CmpFront.add(-1, min, i)$  /* Add dominating nature of  $new$ , front index and solution index */
10:      break
11:    else /*  $new$  and  $F_{min}(i)$  are non-dominating */
12:       $count \leftarrow count + 1$ 
13:    if  $count = |F_{min}|$  then
14:       $CmpFront.add(0, min, 0)$  /* Add dominating nature of  $new$ , front index and solution index */
15:  else
16:     $mid \leftarrow \lfloor (min + max) / 2 \rfloor$  /* Obtain the position of the node to be explored */
17:    for  $i \leftarrow 1$  to  $|F_{mid}|$  do
18:       $isdom \leftarrow \text{domNature}(new, F_{mid}(i))$  /* check the dominating nature of  $new$  and  $F_{mid}(i)$  */
19:      if  $isdom = 1$  then /*  $new$  dominates  $sol$  */
20:         $CmpFront.add(isdom, mid, i)$  /* Add dominating nature of  $new$ , front index and solution index */
21:        if  $mid \neq min$  then /* Check for the existence of left sub-tree */
22:          InsertTree_Right( $\mathcal{F}$ ,  $min$ ,  $mid-1$ ,  $new$ ) /* Explore left sub-tree */
23:        break
24:      else if  $F_{mid}(i) = -1$  then /*  $sol$  dominates  $new$  */
25:         $CmpFront.add(isdom, mid, i)$  /* Add dominating nature of  $new$ , front index and solution index */
26:        InsertTree_Right( $\mathcal{F}$ ,  $mid+1$ ,  $max$ ,  $new$ ) /* Explore right sub-tree */
27:        break
28:      else /*  $new$  and  $F_{mid}(i)$  are non-dominating */
29:         $count \leftarrow count + 1$ 
30:      if  $count = |F_{mid}|$  then /* All solutions in  $F_{mid}$  are non-dominating with  $new$  */
31:         $CmpFront.add(0, mid, 0)$  /* Add dominating nature of  $new$ , front index and solution index */
32:      if  $mid \neq min$  then
33:        InsertTree_Right( $\mathcal{F}$ ,  $min$ ,  $mid-1$ ,  $new$ ) /* Explore left sub-tree */
```

$i \leq len$. In this case it is clear that new will make another front F_{K+1} which is having lower dominance than all the existing K fronts.

4) The $CmpFront$ list is traversed backward i.e. from the end to start. If the two consecutive values denoting dominance nature of new in $CmpFront$ list are different then the position of new is identified otherwise we move to next $CmpFront$ element. Formally for the position of new is to be identified $CmpFront[i]_{dom} \neq CmpFront[i-1]_{dom}$. If this condition is met and if the dominating value of next front (current front is i and next front is $i-1$) is 0 then the new will be inserted in the next front. If dominating value of next front is 1 then $UpdateInsert()$ procedure is called with following parameter - \mathcal{F} , new_{dom} and $CmpFront[i-1]_{fIndex}$.

C. Illustration of InsertTree_Left(\mathcal{F} , min , max , new) procedure

Initially new is compared with the root of the tree. The index of the root is calculated as $mid = \lfloor (1 + K) / 2 \rfloor$.

1. If any solution in the root dominates new then the new will be having lower dominance than the root. So the algorithm explore the right sub-tree of the root using recursive procedure $InsertTree_Left(\mathcal{F}, mid+1, max, new)$.

2. If new and solution in the root is non-dominating then we continue dominance comparison of new with rest of the solutions in the root. If new is non-dominating with rest of the solutions in the root then new can not have lower dominance than root. But it can have higher dominance than the root so for this the left sub-tree is explored using recursive procedure $InsertTree_Left(\mathcal{F}, min, mid-1, new)$.

3. If new dominates the solution in the node then the new has higher dominance than the root node so the left-

sub tree of the root is explored using recursive procedure *InsertTree_Left*(\mathcal{F} , 1, $mid-1$, new).

Terminating Condition of Algorithm 5 and 6: The terminating condition depends on how the index of the root of a sub-tree i.e. mid is calculated. It depends on whether the tree is Left-Balanced or Right-Balanced. The procedure terminates when any one of the following conditions are satisfied.

- 1) A leaf node is encountered.
- 2) If the tree is Left-Balanced Dominance Binary Search Tree and a node with single child is encountered which dominates new .
- 3) If the tree is Right-Balanced Dominance Binary Search Tree and a node with single child is encountered which does not dominate new .

Complexity Analysis: The maximum number of fronts to which new is compared is $\lfloor \log_2 K \rfloor + 1$. Each comparison to the front adds an element in the list *CmpFront*. Thus the space complexity of the proposed algorithm is $\mathcal{O}(\log_2 K)$. In the worst case, the new is to be checked for dominance with all the solutions in the compared fronts. If there is equal division of solutions in all the fronts i.e. each front has $\frac{N}{K}$ solutions then the complexity of the algorithm 5 and 6 is $\mathcal{O}(M \times \frac{N}{K} \times \log K)$ which is $\mathcal{O}(MN)$ is worst case.

D. Complexity Analysis using Dominance Binary Search Tree based Approach

The *InsertTree()* procedure first use either Algorithm 5 or 6 which gives a list known as *CmpFront*. The worst case time complexity of algorithm 5 is $\mathcal{O}(MN)$. After getting *CmpFront*, this list is completely traversed at-most once. In this traversal either a new front is created or *UpdateInsert()* procedure is called. Thus the worst case time complexity of Algorithm 4 is given by $\mathcal{O}(MN) + \mathcal{O}(\log K) + \mathcal{O}(MN^2)$ which is $\mathcal{O}(MN^2)$. The space complexity of dominance tree based approach is $\mathcal{O}(\log K)$ which is used to store the list *CmpFront*.

In case of left dominance binary search tree, the best case occurs when $\mathcal{F} = \{F_1, F_2, F_3\}$ where $n_1 = 1, n_2 = 1, n_3 = N - 2$ and new solution new dominates the solution in both F_1 and F_2 . In this case only two comparisons are required so the best case complexity is $\mathcal{O}(M)$.

In case of right dominance binary search tree, the best case occurs when $\mathcal{F} = \{F_1, F_2\}$ where $n_1 = 1, n_2 = N - 1$ and new solution new dominates the solution in F_1 . In this case only one comparison is needed so the best case complexity is $\mathcal{O}(M)$.

E. Comparison between Linear and Dominance Tree Based Approach

The only difference between the linear and dominance tree based approach is the way to find the index of the front where *UpdateInsert()* procedure is applied. For this purpose, the number of fronts to which new is compared depends on the dominance nature of new with all other solutions in case of linear approach. In case of dominance tree based approach, the number of fronts to which new is compared is either $\lfloor \log_2 K \rfloor$ or $\lfloor \log_2 K \rfloor + 1$. In case of full dominance binary search tree,

the number of fronts to which new is compared is $\lfloor \log_2 K \rfloor + 1$. When the tree is not fully balanced, the number of fronts is either $\lfloor \log_2 K \rfloor$ or $\lfloor \log_2 K \rfloor + 1$ depending on the new .

V. LOOK UP

When a inferior solution is to be removed from \mathcal{F} then first its location should be identified. So in this section we will discuss how to identify the location of a solution in \mathcal{F} . The linear approach proposed in [21], checks for all the fronts in serial manner to identify a given solution. The number of comparison in various scenarios when this technique is followed is described next.

- I. **All Solutions are dominating:** Here all the solutions are dominating in nature i.e. N solutions are divided into N fronts. The maximum number of comparison to search any solution is N . For this the searched solution is in the last front. The minimum comparison to search any solution is 1. Here the searched solution is in the first front.
- II. **All Solutions are non-dominating:** Here all the solutions are non-dominating in nature i.e. all the N solutions are in single front. The maximum number of comparison to search any solution is N . For this the searched solution is the last solution in the front. The minimum time to search any solution is 1. Here the searched solution is the first solution in the front.
- III. **Equal solutions in all the fronts:** When there are equal division of solutions in each front then the number of solutions in each front would be $\frac{N}{K}$. Assume each solution in a front is dominated by all solutions in the preceding front. Maximum number of comparison to search a solution is $K + \frac{N}{K} - 1$. For this the searched solution is the last solution in last front. The minimum number of comparison to search any solution is 1. Here the searched solution is the first solution in the first front.

Two solutions are compared using *CheckDom(A,B)* procedure. The procedure is shown in Algorithm 7. This procedure takes as input two solutions A and B and return the dominance relationship between them. When two solutions are compared then one of the following possibilities may arise:

- Searched solution sol dominates the compared solution in the front. The function returns 1.
- Searched solution sol is dominated by the compared solution in the front. The function returns -1 .
- Searched solution sol is non-dominating with the compared solution in the front. The function returns 0.
- Searched solution sol is same as the compared solution in the front. The function returns 2.

Complexity Analysis: When two solutions are compared then at-most M objectives are to be checked so the worst case complexity of this algorithm is $\mathcal{O}(M)$.

The proposed dominance tree based approach for locating a given solution in \mathcal{F} is described in Algorithm 8. In this algorithm we have used the Left Dominance Binary Search Tree. The corresponding Right Dominance Binary Search Tree

Algorithm 7 *CheckDom*(A, B)

Input: A : First solution, B : Second Solution**Output:** Nature of A with respect to B

```
1: for  $i$  to  $M$  do
2:   if  $A_i < B_i$  then
3:      $flag1 \leftarrow \text{TRUE}$ 
4:   else if  $A_i > B_i$  then
5:      $flag2 \leftarrow \text{TRUE}$ 
6:   else
7:      $count++$ 
8: if  $flag1 = \text{TRUE} \ \&\& \ flag2 = \text{FALSE}$  then
9:   return 1
10: else
11:   if  $flag1 = \text{FALSE} \ \&\& \ flag2 = \text{TRUE}$  then
12:     return -1
13:   else
14:     if  $count = M$  then
15:       return 2
16:     else
17:       return 0
```

can also be used for lookup purpose. Here we first check the searched solution with the solutions in the root of the tree. The procedure *CheckDom*() is used to compare two solutions. When the searched solution is checked with solution in the root front then there are four possibilities:

- 1) If the searched solution dominates any solution in the root then searched solution is having higher dominance than the root so only left sub-tree of root is explored.
- 2) If the searched solution is dominated by any solution in the root then searched solution is having lower dominance than the root so only right sub-tree of root is explored.
- 3) If the searched solution is same as any solution in the root then the lookup process terminates with the index of the front and solution inside the front.
- 4) If searched solution is non-dominating with all the solutions in the root then searched solution can not have lower dominance than root so only left sub-tree of root is explored.

Let the searched solution is same as $F_i(j)$ i.e. the searched solution is the j -th solution in F_i . The algorithm returns (i, j) . In case the solution is not present in \mathcal{F} the algorithm returns -1. The number of comparison in various scenarios when dominance tree based technique is followed is described next.

- I. **All Solutions are dominating:** Here all the solutions are dominating in nature i.e. N solutions are divided into N fronts. The maximum number of comparison to search any solution is given by $\lfloor \log N \rfloor + 1$. For this the searched solution is in the leaf node at depth $\lfloor \log N \rfloor$ (In case of full dominance tree, the searched solution should be at any leaf node because all the leaf node have the same depth and i.e. $\lfloor \log N \rfloor$). The minimum time to search any solution is 1. Here the searched solution is at the mid front.

Algorithm 8 *Lookup*($\mathcal{F}, min, max, sol$)

Input: \mathcal{F} : Set of non-dominated fronts min : Lower index of the front max : Upper index of the front sol : Solution to be searched**Output:** Index of the searched solution

```
1: if  $min = max$  then
2:   for  $i \leftarrow 1$  to  $|F_{min}|$  do
3:      $isDom \leftarrow \text{CheckDom}(sol, F_{min}(i))$ 
4:     if  $isDom = 1$  then
5:       return -1
6:     else if  $isDom = -1$  then
7:       return -1
8:     else if  $isDom = 2$  then
9:       return  $(min, i)$ 
10: else
11:    $mid \leftarrow \lceil \frac{min+max}{2} \rceil$ 
12:   for  $i \leftarrow 1$  to  $|F_{mid}|$  do
13:      $isDom \leftarrow \text{CheckDom}(sol, F_{mid}(i))$ 
14:     if  $isDom = 1$  then
15:       return  $\text{Lookup}(\mathcal{F}, min, mid-1, sol)$ 
16:     else if  $isDom = -1$  then
17:       if  $mid \neq max$  then
18:         return  $\text{Lookup}(\mathcal{F}, mid+1, max, sol)$ 
19:     else if  $isDom = 2$  then
20:       return  $(mid, i)$ 
21:   return  $\text{Lookup}(\mathcal{F}, min, mid-1, sol)$ 
22: return -1
```

- II. **All Solutions are non-dominating:** Here all the solutions are non-dominating in nature i.e. all the N solutions are in single front. The maximum number of comparison to search any solution is given by N . For this the searched solution is the last solution in the front. The minimum time to search any solution is 1. Here the searched solution is the first solution in the front.

- III. **Equal solutions in all the fronts:** When there are equal division of solutions in each front then the number of solutions in each front would be $\frac{N}{K}$. Assume each solution in a front is dominated by all solutions in the preceding front. Maximum number of comparison to search a solution is given by $(\lfloor \log K \rfloor + 1) + \frac{N}{K} - 1 = \lfloor \log K \rfloor + \frac{N}{K}$. For this the searched solution should be the last solution in any leaf node which is at depth $\lfloor \log K \rfloor$ (In case of full dominance tree, for maximum number of comparison, the searched solution should be the last solution in any leaf node. This is because all the leaf node have the same depth and i.e. $\lfloor \log K \rfloor$). The minimum time to search any solution is 1. Here the searched solution is the first solution in the first front.

A. Comparison

When all the solutions are in the same front then both the approaches linear [21] as well as dominance tree based approach perform the same. But in case the number of fronts

are N then the number of comparison vary between 1 to N for linear approach while 1 to $\lfloor \log N \rfloor + 1$ for dominance tree based approach. In this case in some situation linear approach performs better than the dominance tree based approach. Consider an example to illustrate such situation:

Example 5.1: Let there are 100 solutions and these solutions are divided in 100 fronts. In this case the number of comparison for dominance tree based approach vary between 1 to $\lfloor \log 100 \rfloor + 1 = 7$. But in case of linear approach it vary between 1 to 100. Let we want to search the first solution, then the number of comparison using the linear approach is 1 while using dominance tree based approach use 7 comparison.

In general when all the solutions are dominating in nature and if the searched solution is among the first $\lfloor \log N \rfloor + 1$ solutions then the linear approach can outperform the dominance tree based approach otherwise the dominance tree based approach performs better. So when all the solutions are dominating in nature dominance tree based approach performs better for $N - (\lfloor \log N \rfloor + 1)$ solutions and linear approach can perform better for $\lfloor \log N \rfloor + 1$ solutions.

When there is equal division of elements in the fronts i.e. all the fronts have $\frac{N}{K}$ solutions and each solution in a front is dominated by all solutions in the preceding front. Then maximum number of comparison using linear approach would be $K + \frac{N}{K} - 1$ while maximum number of comparison using dominance tree based approach would be $\lfloor \log K \rfloor + \frac{N}{K}$. If the searched solution is among the first $\lfloor \log K \rfloor + 1$ fronts then the linear approach can outperform the tree based approach otherwise the dominance tree based approach performs better. So in this case, dominance tree based approach performs better for $N - \frac{N}{K} (\lfloor \log K \rfloor + 1)$ solutions and linear approach can perform better for $\frac{N}{K} (\lfloor \log K \rfloor + 1)$ solutions.

VI. DELETE A SOLUTION

In this section we will discuss how the structure of the fronts changes after the removal of a solution sol from the set of fronts. Algorithm 9 shows the deletion procedure of a solution sol in the list of non-dominated fronts \mathcal{F} . This algorithm does not run the complete sorting algorithm again. It uses the non-dominance properties of the solution in the same front and uses the ranking of the front. Here we are assuming that all the fronts are arranged in decreasing order of their dominance.

First of all the position of the deleted solution is to be identified in \mathcal{F} . The position (i, j) refers that the deleted solution is j -th solution in i -th front. This operation is carried out using either linear search or *LookUp()* procedure which is described in detail in Section V.

After the identification of the deleted solution, the solution is deleted from the front F_i . The removal of a solution from front F_i requires re-arrangement of solutions in the fronts $F_i, F_{i+1}, F_{i+2}, \dots, F_K$. If the deleted solution is from the last front i.e. F_K then no solution is re-arranged and the process of deletion terminates. But if the deleted solution is in front $F_k, 1 \leq k < K$ then the re-arrangement of solutions occurs. This re-arrangement is performed by *UpdateDelete()* procedure which is described in Algorithm 10.

Algorithm 9 Delete(\mathcal{F}, sol)

Input: $\mathcal{F} = \{F_1, F_2, \dots, F_K\}$: Non-dominated fronts in the decreasing order of their dominance
new: A new solution

Output: Updated \mathcal{F} after removal of sol

- 1: $(i, j) \leftarrow \text{LookUp}(\mathcal{F}, sol)$
 - 2: $F_i \leftarrow F_i \setminus F_i(j)$
 - 3: **if** $i \neq K$ **then**
 - 4: $\text{UpdateDelete}(\mathcal{F}, i)$
-

Algorithm 10 UpdateDelete($\mathcal{F}, index$)

Input: \mathcal{F} : Set of non-dominated fronts

index: Non-domination level index

Output: Updated \mathcal{F}

- 1: $l \leftarrow |F_{index}|$
 - 2: **for** $i \leftarrow 1$ to $|F_{index+1}|$ **do**
 - 3: $count \leftarrow 0$
 - 4: **for** $j \leftarrow 1$ to l **do**
 - 5: **if** $\text{dominates}(F_{index+1}(i), F_{index}(j)) = 0$ **then**
 - 6: $count \leftarrow count + 1$
 - 7: **if** $count = l$ **then**
 - 8: $F_{index} \leftarrow F_{index} \cup F_{index+1}(i)$
 - 9: $F_{index+1} \leftarrow F_{index+1} \setminus F_{index+1}(i)$
 - 10: $i \leftarrow i - 1$
 - 11: **if** $F_{index+1} = \Phi$ **then**
 - 12: Decrease the domination level of front $F_k, k \in \{index+2, index+3, \dots, K\}$
 - 13: **else if** $|F_{index}| = l$ **then**
 - 14: // Do nothing process terminates
 - 15: **else**
 - 16: $\text{UpdateDelete}(\mathcal{F}, index+1)$
-

A. Illustration of UpdateDelete(\mathcal{F}, i) procedure

This procedure takes as input the set of non-dominated fronts \mathcal{F} and the index of the front F_i from where the solution is deleted. This procedure updates \mathcal{F} by either removing an existing front or by re-arranging the solutions within the existing fronts.

Initially the cardinality of F_{index} is stored in l . This is because when the solutions from $F_{index+1}$ is compared with F_{index} for non-dominance then it should be compared with only first l solutions. Here we find the solutions in $F_{index+1}$ which are non-dominated with F_{index} . The solutions which are non-dominated with F_{index} are added to it and removed from $F_{index+1}$. This removal guarantees that no solution occupy more than one place. After this addition and removal, following cases can occur:

- If all the solutions in front $F_{index+1}$ are merged to front F_{index} i.e. $F_{index+1} = \Phi$ then the non-domination level of fronts $F_{index+2}, F_{index+3}, \dots, F_K$ are decreased by 1.
- When no solution from front $F_{index+1}$ is merged with F_{index} i.e. $|F_{index}| = l$ then the process terminates.
- If some of the solutions in front $F_{index+1}$ are merged

to front F_{index} i.e. $F_{index+1} \neq \Phi$ then the procedure is repeated with $UpdateDelete(\mathcal{F}, index+1)$.

Complexity Analysis: In this algorithm the maximum number of comparison is performed when the deleted solution is from the front F_1 i.e. the procedure is called with $index = 1$. The maximum number of comparison occurs when each call to this procedure shifts one solution in higher level front after comparing with all the solutions. Thus the maximum number of comparison for this procedure is given by $(n_1-1)n_2 + (n_2-1)n_3 + \dots + (n_{K-1}-1)n_K$. Each comparison between two solutions requires at-most M comparison between M objectives. Thus the worst case complexity of this procedure is $\mathcal{O}(MN^2)$.

B. Complexity of Proposed Approach

Here we will analyze the complexity of the proposed approach. We can see from all the algorithms 8, 9 and 10 that they involve scalar variables only except for the given set of fronts $F_k, 1 \leq k \leq K$, where K is the number of fronts. Therefore, the space complexity of the proposed approach is $\mathcal{O}(1)$. The worst case time complexity of $UpdateDelete()$ procedure dominates the worst case time complexity of $LookUp()$ procedure. The complexity of $UpdateDelete()$ procedure is quadratic while $LookUp()$ has linear complexity. Thus the overall worst case complexity of the proposed approach is $\mathcal{O}(MN^2)$.

When sequential search strategy is used the the best case occurs when F_1 has single solution which is to be deleted. In this case only one comparison is required so the best case complexity is $\mathcal{O}(M)$.

When dominance tree based search strategy is used the the best case occurs when F_{mid} has single solution which is to be deleted. In this case only one comparison is required so the best case complexity is $\mathcal{O}(M)$.

VII. NUMBER OF DOMINANCE COMPARISON

In this section, we obtain the maximum number of dominance comparison occurred while inserting a new solution or deleting an existing solution in given set of fronts using the linear as well as dominance tree based approach. In case of deletion, the linear approach uses the sequential search for locating the solution while tree based approach uses the $LookUp()$ procedure for the same.

A. Linear Approach

Here we obtain the maximum number of dominance comparison when either a new solution new is being inserted in \mathcal{F} or an existing solution sol is being deleted from \mathcal{F} using linear approach.

1) *Insert:* For maximum number of dominance comparison in linear approach, the new solution new dominates n_1-1 solutions in the first front. So maximum number of dominance comparison is given by Equation 1. In this case the $UpdateInsert()$ procedure is called with index value 2.

$$\#Comp_{Linear} = n_1 + [(n_1-1) \cdot n_2 + \dots + (n_{K-1}-1) \cdot n_K] \quad (1)$$

$\#Comp_{Linear}$ attains its maximum value when there are exactly two fronts. For proof see Appendix A. In case of even number of solutions, first front should have $\frac{N}{2}+1$ solutions while second front should have $\frac{N}{2}-1$ solutions. In case of odd number of solutions, the first front has $\lceil \frac{N}{2} \rceil$ solutions and second front has $\lfloor \frac{N}{2} \rfloor$. In this way the maximum number of comparisons is

N is Even:

$$\#Comp_{Linear} = (\frac{N}{2}+1) + [(\frac{N}{2}+1)-1] (\frac{N}{2}-1) = \frac{N^2}{4} + 1$$

N is Odd:

$$\begin{aligned} \#Comp_{Linear} &= \lceil \frac{N}{2} \rceil + [\lceil \frac{N}{2} \rceil - 1] \lfloor \frac{N}{2} \rfloor \\ &= \frac{N+1}{2} + [\frac{N+1}{2} - 1] \frac{N-1}{2} = \frac{N^2+3}{4} = \lceil \frac{N^2}{4} \rceil \end{aligned}$$

$$\#Comp_{Linear} = \begin{cases} \frac{N^2}{4} + 1 & \text{if } N \text{ is even} \\ \lceil \frac{N^2}{4} \rceil & \text{if } N \text{ is odd} \end{cases}$$

2) *Delete:* For maximum number of dominance comparison the value of Equation 1 should be maximized. Here the deleted solution sol is the last solution in the first front. So the $UpdateDelete()$ procedure is called with index value 1.

B. Left Dominance Binary Search Tree Based Approach

Here the maximum number of dominance comparison in case of insertion and deletion of a solution is obtained when Left Dominance Binary Search Tree based approach is used.

1) *Insert:* The index of the root front is obtained by $mid = \lceil \frac{1+N}{2} \rceil$. The height of the dominance tree $h = \lfloor \log N \rfloor$. For maximum number of dominance comparison, the new solution new dominates the $n_1 - 1$ solutions in the first front. So, maximum number of dominance comparison is given by Equation 2.

$$\begin{aligned} \#Comp_{LeftTree} &= [n_{\lceil \frac{mid}{2^0} \rceil} + n_{\lceil \frac{mid}{2^1} \rceil} + n_{\lceil \frac{mid}{2^2} \rceil} + \dots + n_{\lceil \frac{mid}{2^h} \rceil}] + \\ &[(n_1-1) \cdot n_2 + (n_2-1) \cdot n_3 + \dots + (n_{K-1}-1) \cdot n_K] \quad (2) \end{aligned}$$

This value will be maximum when there are exactly two fronts. For proof see Appendix B. In case of even number of solutions, first front should have $\frac{N}{2} + 1$ solutions while second front should have $\frac{N}{2} - 1$ solutions. In case of odd number of solutions, the first front has $\lceil \frac{N}{2} \rceil$ solutions and second front has $\lfloor \frac{N}{2} \rfloor$. In this way the maximum number of comparison is

N is Even:

$$\begin{aligned} \#Comp_{LeftTree} &= (\frac{N}{2}-1) + (\frac{N}{2}+1) + [(\frac{N}{2}+1-1) (\frac{N}{2}-1)] \\ &= \frac{N^2}{4} + \frac{N}{2} \end{aligned}$$

N is Odd:

$$\begin{aligned} \#Comp_{LeftTree} &= \lfloor \frac{N}{2} \rfloor + \lceil \frac{N}{2} \rceil + [(\lceil \frac{N}{2} \rceil - 1) \lfloor \frac{N}{2} \rfloor] \\ &= N + [(\frac{N+1}{2} - 1) \frac{N-1}{2}] \\ &= \frac{N^2}{4} + \frac{N}{2} + \frac{1}{4} \end{aligned}$$

2) *Delete:* For maximum number of dominance comparison, the deleted solution sol has to be the last solution in first front F_1 . So, maximum number of dominance comparison is given by Equation 2. Here the deleted solution is located using left dominance binary search tree.

C. Right Dominance Binary Search Tree Based Approach

Here the maximum number of dominance comparison in case of insertion and deletion of a solution is obtained when Right Dominance Binary Search Tree based approach is used.

1) *Insert*: The index of the root front is obtained by $mid = \lfloor \frac{1+N}{2} \rfloor$. The height of the dominance tree $h = \lfloor \log N \rfloor$. For maximum number of dominance comparison, the new solution *new* dominates the solutions in the first front. So, maximum number of dominance comparison is given by Equation 3.

$$\#Comp_{RightTree} = \left[n_{\lfloor \frac{mid}{2^0} \rfloor} + n_{\lfloor \frac{mid}{2^1} \rfloor} + \dots + n_{\lfloor \frac{mid}{2^h} \rfloor} \right] + [(n_1 - 1) \cdot n_2 + (n_2 - 1) \cdot n_3 + \dots + (n_{K-1} - 1) \cdot n_K] \quad (3)$$

This value will be maximum when there are exactly two fronts. For proof see Appendix C. In case of even number of solutions, first front should have $\frac{N}{2} + 1$ solutions while second front should have $\frac{N}{2} - 1$ solutions. In case of odd number of solutions, the first front has $\lceil \frac{N}{2} \rceil$ solutions and second front has $\lfloor \frac{N}{2} \rfloor$. In this way the maximum number of comparison is **N is Even:**

$$\#Comp_{RightTree} = \left(\frac{N}{2} + 1 \right) + \left[\left(\frac{N}{2} + 1 - 1 \right) \left(\frac{N}{2} - 1 \right) \right] = \frac{N^2}{4} + 1$$

N is Odd:

$$\begin{aligned} \#Comp_{LeftTree} &= \lceil \frac{N}{2} \rceil + \left[\left(\lceil \frac{N}{2} \rceil - 1 \right) \left\lfloor \frac{N}{2} \right\rfloor \right] \\ &= \frac{N+1}{2} + \left[\left(\frac{N+1}{2} - 1 \right) \frac{N-1}{2} \right] \\ &= \frac{N^2+3}{4} = \lceil \frac{N^2}{4} \rceil \end{aligned}$$

2) *Delete*: For maximum number of dominance comparison, the deleted solution *sol* has to be the last solution in the first front F_1 . So, maximum number of dominance comparison is given by Equation 3. Here the deleted solution is located using right dominance binary search tree.

VIII. CASE STUDY: ALL SOLUTIONS ARE NON-DOMINATED

In this section, we will discuss the maximum and minimum number of dominance comparison needed when either a new solution is inserted or an existing solution is deleted from the set of fronts where there is single front containing all the N solutions. In this case the linear as well as dominance tree based approach performs the same.

A. Insert

The maximum and minimum number of dominance comparison is discussed here in case of insertion. The value of $K = 1$. Table IIa shows this scenario. The changed structure in the front after insertion is also shown.

Maximum Number of Dominance Comparison: The maximum number of dominance comparison is N . Here *new* is compared with all the solutions in the front. In this case there are four possibilities.

- I. *new* will be merged to the front if it is non-dominating with each solution in the front. This is shown in Table IIb.
- II. *new* makes another front having lower dominance (lower rank) than current front if it is non-dominating with first

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_N$

(a)

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_N, new$

(b)

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_N$
F_2	<i>new</i>

(c)

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_N, new$
F_2	$sol_{p+1}, sol_{p+2}, \dots, sol_N$

(d)

\mathcal{F}	Solutions
F_1	<i>new</i>
F_2	$sol_1, sol_2, \dots, sol_N$

(e)

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_N$
F_2	<i>new</i>

(f)

\mathcal{F}	Solutions
F_1	$sol_1, sol_2, \dots, sol_{N-1}$

(g)

\mathcal{F}	Solutions
F_1	sol_2, \dots, sol_N

(h)

TABLE II: Initial and changed structure of fronts using linear approach when all the solutions are non-dominating.

$N - 1$ solutions and is dominated by the N -th solution. See Table IIc for this.

- III. *new* merges in the same front and the solutions which are dominated by *new* make another front having lower dominance than current front. This is shown in Table IId.
- IV. *new* makes another front having higher dominance than the current front. This is possible when *new* dominates all the solutions in the front. See Table IIf.

Minimum Number of Dominance Comparison: The minimum number of dominance comparison is 1. Here *new* is compared with only first solution in the front. *new* makes another front having lower dominance than the current front because *new* is dominated by first solution in the front. Refer Table IIe.

B. Delete

The maximum and minimum number of dominance comparison is discussed here in case of deletion of an existing solution. The changed structure in the front after insertion is also shown.

Maximum Number of Dominance Comparison: Maximum number of comparison occurs when the last solution i.e. sol_N is deleted from the front. Table IIg shows this scenario.

Minimum Number of Dominance Comparison: Minimum number of comparison occurs when the first solution i.e. sol_1 is deleted from the front. Table IIh shows this scenario.

IX. CASE STUDY: ALL SOLUTIONS ARE DOMINATED

In this section, we will discuss the maximum and minimum number of dominance comparison needed when either a new solution is inserted or an existing solution is deleted from the set of fronts where there are N fronts. Each front contains single solution.

A. Linear

The number of comparison using linear approach is discussed.

1) *Insert*: The maximum and minimum number of dominance comparison is discussed here in case of insertion. The value of $K = N$. Table IIIa shows this scenario. The changed structure in the front after insertion is also shown.

Maximum Number of Dominance Comparison: The maximum number of dominance comparison is N . Here *new* will be compared with solution in each front. In this case there are two possibilities.

- I. *new* will be merged to the last front if *new* is dominated by the solutions in the first $N - 1$ fronts and it is non-dominating with the solution in the last front. See Table IIIb.
- II. *new* makes another front if *new* is dominated by the solutions in all the N fronts. The dominance of *new* will be the lowest among all the fronts. Refer Table IIIc.

Minimum Number of Dominance Comparison: When all the solutions belong to different front then the minimum number of dominance comparison will be 1. Here *new* will be compared with only single solution. In this case there are two possibilities.

- I. *new* will be merged to the first front if *new* is non-dominating with the solution in the first front. Refer Table IIId.
- II. *new* makes another front (having higher dominance than the first front) if it dominates the solution in the first front. See Table IIIf.

Front	Solutions	Front	Solutions	Front	Solutions	Front	Solutions
F_1	sol_1	F_1	sol_1	F_1	sol_1	F_1	sol_1
F_2	sol_2	F_2	sol_2	F_2	sol_2	F_2	sol_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
F_N	sol_N	F_N	sol_N, new	F_N	sol_N	F_N	sol_N
				F_{N+1}	new		

(a) (b) (c) (d)

Front	Solutions	Front	Solutions	Front	Solutions
F_1	new	F_1	sol_1	F_1	sol_2
F_2	sol_1	F_2	sol_2	F_2	sol_3
F_3	sol_2	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	F_{N-1}	sol_{N-1}	F_{N-1}	sol_N
F_{N+1}	sol_N				

(e) (f) (g)

TABLE III: Initial and changed structure of fronts using linear approach when all the solutions are dominating.

2) *Delete*: The maximum and minimum number of dominance comparison is discussed here in case of deletion of an existing solution. The changed structure in the front after insertion is also shown.

Maximum Number of Dominance Comparison: The maximum number of dominance comparison will be N . This occurs when the deleted solution is in the last front i.e. sol_N is being deleted from the front. Table IIIg shows this scenario.

Minimum Number of Dominance Comparison: The minimum number of dominance comparison will be 1 and it occurs when the deleted solution is in the first front i.e. sol_1 is being deleted from the front. Table IIIg shows this scenario.

Front	Solutions	Front	Solutions	Front	Solutions
F_1	sol_1	F_1	sol_1	F_1	sol_1
F_2	sol_2	F_2	sol_2	F_2	sol_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
F_p	sol_p, new	F_p	sol_p	F_p	sol_p
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
F_N	sol_N	F_N	sol_N	F_{N+1}	new

(a) (b) (c)

Front	Solutions	Front	Solutions
F_1	new	F_1	sol_1
F_2	sol_1	\vdots	\vdots
F_3	sol_2	\vdots	\vdots
\vdots	\vdots	F_p	sol_p
F_{N+1}	sol_N	F_{p+1}	new
		\vdots	\vdots
		F_{N+1}	sol_N

(d) (e)

TABLE IV: Initial and changed structure of fronts using linear approach when all the solutions are dominating.

B. Dominance Tree Based Approach

The number of comparison using dominance tree based approach is discussed.

1) *Insert*: The maximum and minimum number of dominance comparison is discussed here in case of insertion. The value of $K = N$. Table IVa shows this scenario. The changed structure in the front after insertion is also shown. In this case the number of dominance comparison will be $\lfloor \log_2 N \rfloor + 1$ in all the cases. Here *new* will be compared with solution in $\lfloor \log_2 N \rfloor + 1$ front (there is only one solution in each front). In this case there are two possibilities.

- I. *new* will be merged to any front. See Table IVb.
- II. *new* makes another front.
 - i. This new front can have lower dominance than the last front. See Table IVc.
 - ii. This new front can have higher dominance than the first front. See Table IVd.
 - iii. This new front can have dominance in between the first and the last front. See Table IVe.

2) *Delete*: The maximum and minimum number of dominance comparison is discussed here in case of deletion. The value of $K = N$.

Maximum Number of Dominance Comparison: The maximum number of dominance comparison will be $\lfloor \log_2 N \rfloor + 1$. Maximum number of comparison occurs when the deleted solution is at the leaf of the tree.

Minimum Number of Dominance Comparison: The minimum number of dominance comparison will be 1. Minimum number of comparison occurs when the deleted solution is at the root of the tree.

Table V shows the comparison among different approaches when considered for non-domination level update problem. Most of the approach perform complete sorting algorithm when either an insertion or a deletion occurs. This table also shows the best and worst case time complexity in case of insertion/deletion. This table also shows the maximum number of comparison in case of insertion/deletion. The worst case space complexity of all the approaches are also shown.

Approach	Space Complexity: Worst Case	Time Complexity		Maximum Number of comparison	
		Best Case	Worst Case	Insert	Delete
Naive approach	$\mathcal{O}(N)^*$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^3)$	$\frac{N(N+1)(N+2)}{6}$	$\frac{(N-2)(N-1)N}{6}$
Fast Non-dominated Sort	$\mathcal{O}(N^2)^*$	$\mathcal{O}(MN^2)$	$\mathcal{O}(MN^2)$	$(N+1)N$	$(N-1)(N-2)$
Climbing Sort	$\mathcal{O}(N^2)^*$		$\mathcal{O}(MN^2)$	$(N+1)N$	$(N-1)(N-2)$
Deductive Sort	$\mathcal{O}(N)^*$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$	$\frac{(N+1)N}{2}$	$\frac{(N-1)(N-2)}{2}$
Arena's Sort	$\mathcal{O}(N)^*$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$	$\frac{(N+1)N}{2}$	$\frac{(N-1)(N-2)}{2}$
ENS-SS	$\mathcal{O}(1)^*$	$\mathcal{O}(MN\sqrt{N})$	$\mathcal{O}(MN^2)$	$\frac{(N+1)N}{2}$	$\frac{(N-1)(N-2)}{2}$
ENS-BS	$\mathcal{O}(1)^*$	$\mathcal{O}(MN \log N)$	$\mathcal{O}(MN^2)$	$\frac{(N+1)N}{2}$	$\frac{(N-1)(N-2)}{2}$
ENLU	$\mathcal{O}(N)^*$	$\mathcal{O}(M)$	$\mathcal{O}(MN^2)$	N Even: $\frac{N^2}{4} + 1$	N Even: $\frac{N^2}{4} + 1$
				N Odd: $\lceil \frac{N^2}{4} \rceil$	N Odd: $\lceil \frac{N^2}{4} \rceil$
Linear	$\mathcal{O}(1)^*$	$\mathcal{O}(M)$	$\mathcal{O}(MN^2)$	N Even: $\frac{N^2}{4} + 1$	N Even: $\frac{N^2}{4} + 1$
				N Odd: $\lceil \frac{N^2}{4} \rceil$	N Odd: $\lceil \frac{N^2}{4} \rceil$
Left Dominance Tree	Insert: $\mathcal{O}(\log N)$	$\mathcal{O}(M)$	$\mathcal{O}(MN^2)$	N Even: $\frac{N^2}{4} + \frac{N}{2}$	N Even: $\frac{N^2}{4} + \frac{N}{2}$
	Delete: $\mathcal{O}(1)$			N Odd: $\frac{N^2}{4} + \frac{N}{2} + \frac{1}{4}$	N Odd: $\frac{N^2}{4} + \frac{N}{2} + \frac{1}{4}$
Right Dominance Tree	Insert: $\mathcal{O}(\log N)$	$\mathcal{O}(M)$	$\mathcal{O}(MN^2)$	N Even: $\frac{N^2}{4} + 1$	N Even: $\frac{N^2}{4} + 1$
	Delete: $\mathcal{O}(1)$			N Odd: $\lceil \frac{N^2}{4} \rceil$	N Odd: $\lceil \frac{N^2}{4} \rceil$

TABLE V: Space and Time complexities of the approaches when considered for non-domination level update problem. * shows that the worst case space complexity for insertion/deletion are same.

X. SORTING

In this section, we will discuss our proposed non-dominating sorting algorithm. For this purpose, we can use either linear or dominance binary search tree based approach discussed in above section. The process of sorting is described in Algorithm 11. This sorting algorithm is incremental in nature which means this algorithm does not require all the solutions beforehand. This algorithm sorts the solutions as they arrive. So this algorithm can also be used as on-line algorithm [27], [28] because it sorts the solutions as they arrive.

In this algorithm, initially first solution is inserted into the front and there was no solution in the front so the first solution to be added directly to the front. This solution alone is sorted. Then second solution is added to the front. After the insertion of the second solution, the two solutions are in sorted from. After this the third solution is being inserted in the sorted front and this same process continues for all the solutions.

Competitive Ratio: The performance of an on-line algorithm is evaluated by Competitive Ratio [29]. As defined in [29], the competitive ratio of an on-line algorithm over all possible input sequence is the ratio between the cost incurred by on-line algorithm and the cost incurred by optimal off-line algorithm. Thus an optimal on-line algorithm is one whose competitive ratio is less. An on-line algorithm is known to be competitive if its competitive ratio is bounded.

Let the on-line algorithm be *ONSort* and the corresponding optimal off-line algorithm be *OFFSort*. Let the sequence of solutions to be sorted is \mathbb{S} . The cost incurred using *ONSort* is *ONSort*(\mathbb{S}) and using *OFFSort* is *OFFSort*(\mathbb{S}). Algorithm *ONSort* is known to be k -competitive if there exists a constant c such that *ONSort*(\mathbb{S}) $\leq k \cdot \text{OFFSort}(\mathbb{S}) + c$ for all sequence of solution. Also there should be no relation between the c and the input sequence \mathbb{S} .

There are various off-line algorithm proposed for non-dominating sorting e.g. Fast Non-dominated Sort [9], Climbing

Sort [15], Deductive Sort [15], Arena's Sort [24], ENS-SS [16], ENS-BS [16]. In the worst case, the time complexity of each algorithm is $\mathcal{O}(MN^2)$. Thus the worst case complexity of optimal off-line algorithm for non-dominated sorting is $\mathcal{O}(MN^2)$. The worst case time complexity of the on-line sorting algorithm is given by $\mathcal{O}(MN^3)$ because the insertion of single solution takes $\mathcal{O}(MN^2)$ time and there are N such solutions. Thus the competitive ratio of the on-line algorithm for sorting is given by N . The following relation holds *ONSort*(\mathbb{S}) $\leq N \cdot \text{OFFSort}(\mathbb{S})$

Algorithm 11 Sorting(P)

Input: Population: P

Output: \mathcal{F} : All non-dominated fronts of P in increasing order of their ranks

- 1: $\mathcal{F} \leftarrow \text{null}$
 - 2: **for each** $\text{sol} \in P$ **do**
 - 3: **if** $\mathcal{F} = \text{null}$ **then**
 - 4: All sol to \mathcal{F}
 - 5: **else**
 - 6: $\text{InsertLinear}(\mathcal{F}, \text{sol})$
-

XI. CONCLUSION & FUTURE WORK

In this paper we have proposed the modified version of ENLU approach which is efficient in terms of space. In this paper we have also proposed the new approach based on dominance tree based technique to solve Non-domination Level Update problem. Two variants of this tree are discussed and the update problem can be solved using both the types of tree. This technique inserts the new solution to its correct position and update the dominance level of those solutions which are to be updated. The technique to delete inferior solution is also described. To identify the correct position of the deleted solution dominance tree based approach is used.

The maximum number of possible comparisons required in either inserting a solution in the set of fronts or deleting a solution is also obtained. The behaviour of the approach for some special cases are also analysed. At the end, using the proposed technique for Non-domination Level Update problem, a sorting algorithm is provided which does not require all the solutions beforehand unlike all other existing algorithm [9], [15], [16]. So the proposed sorting algorithm can be used where all the solutions are not known in advance. This algorithm is on-line so the competitive ratio of this algorithm is proven to be N .

In future we would like to minimize the number of dominance comparison in the situation when a solution is being either inserted or deleted. In this paper we have used the tree structure for the set of fronts and the solutions inside the fronts are considered in linear manner. It would be interesting to see whether the tree structure in the fronts can improve the number of dominance comparison as done in [22], [30].

REFERENCES

- [1] T. Bäck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.
- [2] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.
- [3] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC Press, 2000, vol. 1.
- [4] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artificial intelligence*, vol. 85, no. 1, pp. 245–276, 1996.
- [5] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.
- [6] C. A. C. Coello, D. A. Van Veldhuizen, and G. B. Lamont, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2002, vol. 242.
- [7] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," *Lecture notes in computer science*, vol. 1917, pp. 849–858, 2000.
- [8] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Shaker Ithaca, 1999, vol. 63.
- [9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.
- [10] E. Zitzler, M. Laumanns, L. Thiele, E. Zitzler, L. Thiele, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," 2001.
- [11] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Evolutionary computation*, vol. 8, no. 2, pp. 149–172, 2000.
- [12] D. W. Corne, J. D. Knowles, and M. J. Oates, "The pareto envelope-based selection algorithm for multiobjective optimization," in *Parallel Problem Solving from Nature PPSN VI*. Springer, 2000, pp. 839–848.
- [13] H. A. Abbass, R. Sarker, and C. Newton, "Pde: a pareto-frontier differential evolution approach for multi-objective optimization problems," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2. IEEE, 2001, pp. 971–978.
- [14] A. Berry and P. Vamplew, "The combative accretion model—multiobjective optimisation without explicit pareto ranking," in *Evolutionary Multi-Criterion Optimization*. Springer, 2005, pp. 77–91.
- [15] K. McClymont and E. Keedwell, "Deductive sort and climbing sort: New methods for non-dominated sorting," *Evolutionary computation*, vol. 20, no. 1, pp. 1–26, 2012.
- [16] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "An efficient approach to nondominated sorting for evolutionary multiobjective optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 19, no. 2, pp. 201–213, 2015.
- [17] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," *Addison wesley*, vol. 1989, 1989.
- [18] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.
- [19] K. Deb, M. Mohan, and S. Mishra, "Evaluating the ϵ -domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions," *Evolutionary computation*, vol. 13, no. 4, pp. 501–525, 2005.
- [20] N. Beume, B. Naujoks, and M. Emmerich, "Sms-emoa: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.
- [21] K. Li, K. Deb, Q. Zhang, and S. Kwong, "Efficient non-domination level update approach for steady-state evolutionary multiobjective optimization," *Department of Electrical and Computer Engineering, Michigan State University, East Lansing, USA, Tech. Rep. COIN Report*, no. 2014014, 2014.
- [22] I. Yakupov and M. Buzdalov, "Incremental non-dominated sorting with $o(n)$ insertion for the two-dimensional case," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 1853–1860.
- [23] M. Buzdalov, I. Yakupov, and A. Stankevich, "Fast implementation of the steady-state nsga-ii algorithm for two dimensions based on incremental non-dominated sorting," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2015.
- [24] S. Tang, Z. Cai, and J. Zheng, "A fast method of constructing the non-dominated set: arena's principle," in *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, vol. 1. IEEE, 2008, pp. 391–395.
- [25] J. Vuillemin, "A unifying look at data structures," *Communications of the ACM*, vol. 23, no. 4, pp. 229–239, 1980.
- [26] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.
- [27] R. M. Karp, "On-line algorithms versus off-line algorithms: How much is it worth to know the future?" in *IFIP Congress (I)*, vol. 12, 1992, pp. 416–429.
- [28] S. Albers, "Online algorithms: a survey," *Mathematical Programming*, vol. 97, no. 1-2, pp. 3–26, 2003.
- [29] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.
- [30] M. Buzdalov and V. Parfenov, "Various degrees of steadiness in nsga-ii and their influence on the quality of results," in *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*. ACM, 2015, pp. 749–750.

APPENDIX A

LINEAR APPROACH

The maximum number of comparisons in case of linear approach is given by

$$f_{linear} = n_1 + [(n_1-1) \cdot n_2 + (n_2-1) \cdot n_3 + \dots + (n_{K-1}-1) \cdot n_K]$$

Now our aim is to obtain the maximum value of f_{linear} so that the maximum number of dominance comparisons can be obtained. We obtain this in following manner.

A. Number of fronts is 2

The population \mathbb{P} of size N is divided in two fronts i.e. $\mathcal{F} = \{F_1, F_2\}$. Let $|F_1| = n_1$ and $|F_2| = n_2$ so $N = n_1 + n_2$.

$$\begin{aligned} f_{linear} &= n_1 + (n_1-1) n_2 \\ &= n_1 + (n_1-1) (N-n_1) \\ &= n_1 N + 2n_1 - n_1^2 - N \\ \frac{df_{linear}}{dn_1} &= N + 2 - 2n_1 \\ \frac{d^2 f_{linear}}{dn_1^2} &= -2 \end{aligned}$$

The maximum value of f_{linear} is achieved when $\frac{df_{linear}}{dn_1} = 0$ and $\frac{d^2 f_{linear}}{dn_1^2} < 0$. Thus using this equality we get, $n_1 = \frac{N}{2} + 1$ and $n_2 = \frac{N}{2} - 1$. The maximum value of f_{linear} is as follows:

$$\begin{aligned} f_{linear} &= \left(\frac{N}{2} + 1\right) + \left[\left(\frac{N}{2} + 1\right) - 1\right] \left(\frac{N}{2} - 1\right) \\ &= \frac{N^2}{4} + 1 \end{aligned}$$

B. Number of fronts is 3

The population \mathbb{P} of size N is divided in three fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3\}$. Let $|F_1| = n_1$, $|F_2| = n_2$ and $|F_3| = n_3$ so $N = n_1 + n_2 + n_3$.

$$\begin{aligned} f_{linear} &= n_1 + (n_1 - 1)n_2 + (n_2 - 1)n_3 \\ &= n_1 n_2 + n_2 n_3 + 2n_1 - N \\ &= (N - n_2 - n_3)n_2 + n_2 n_3 + 2(N - n_2 - n_3) - N \\ &= Nn_2 + N - n_2^2 - 2n_2 - 2n_3 \end{aligned}$$

The value of f_{linear} will be maximized when $n_3 = 0$. Thus $N = n_1 + n_2$. From the subsection A-A, we can conclude that when the population is divided into 2 fronts then the maximum number of dominance comparison will be $\frac{N^2}{4} + 1$. Hence for maximum number of dominance comparison, $n_1 = \frac{N}{2} + 1$, $n_2 = \frac{N}{2} - 1$, $n_3 = 0$.

C. Number of fronts is 4

The population \mathbb{P} of size N is divided in four fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$. $|F_1| = n_1$, $|F_2| = n_2$, $|F_3| = n_3$ and $|F_4| = n_4$ so $N = n_1 + n_2 + n_3 + n_4$.

$$\begin{aligned} f_{linear} &= n_1 + (n_1 - 1)n_2 + (n_2 - 1)n_3 + (n_3 - 1)n_4 \\ &= n_1 n_2 + n_2 n_3 + n_3 n_4 + 2n_1 - N \\ &= n_2(n_1 + n_3) + n_3 n_4 + 2n_1 - N \\ &= n_2(N - n_2 - n_4) + n_3(N - n_1 - n_2 - n_3) + 2n_1 - N \\ &= (n_2 + n_3)N - (n_2^2 + n_3^2) - (n_1 + n_2)n_3 + 2n_1 - n_2 n_4 - N \end{aligned}$$

The value of f_{linear} will be maximized when $n_4 = 0$. Thus $N = n_1 + n_2 + n_3$. From the subsection A-B, we can conclude that when the population is divided into 3 fronts then the maximum value of the function is $\frac{N^2}{4} + 1$. Thus For maximum number of dominance comparison, $n_1 = \frac{N}{2} + 1$, $n_4 = \frac{N}{2} - 1$, $n_3 = 0$, $n_4 = 0$.

D. Number of fronts is 5

The population \mathbb{P} of size N is divided in five fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3, F_4, F_5\}$. $|F_1| = n_1$, $|F_2| = n_2$, $|F_3| = n_3$, $|F_4| = n_4$ and $|F_5| = n_5$ so $N = n_1 + n_2 + n_3 + n_4 + n_5$.

$$\begin{aligned} f_{linear} &= n_1 + (n_1 - 1)n_2 + (n_2 - 1)n_3 + (n_3 - 1)n_4 + (n_4 - 1)n_5 \\ &= n_1 n_2 + n_2 n_3 + n_3 n_4 + n_4 n_5 + 2n_1 - N \\ &= n_1 n_2 + n_2(N - n_1 - n_2 - n_4 - n_5) + (N - n_1 - n_2 - n_4 - n_5)n_4 + n_4 n_5 + 2n_1 - N \\ &= (n_2 + n_4)N - (n_2^2 + n_4^2) - 2n_2 n_4 - n_1 n_4 + 2n_1 - N - n_2 n_5 \end{aligned}$$

The value of f_{linear} will be maximized when $n_5 = 0$. Thus $N = n_1 + n_2 + n_3 + n_4$. From the subsection A-C, we can conclude that when the population is divided into 4 fronts then the maximum value of the function is $\frac{N^2}{4} + 1$. Hence For maximum number of dominance comparison, $n_1 = \frac{N}{2} + 1$, $n_2 = \frac{N}{2} - 1$, $n_3 = 0$, $n_4 = 0$, $n_5 = 0$.

Thus we can conclude that the maximum number of comparisons in case of linear approach is $\frac{N^2}{4} + 1$ which occurs when there are two fronts i.e. $\mathcal{F} = \{F_1, F_2\}$. The cardinality of each front is $|F_1| = \frac{N}{2} + 1$ and $|F_2| = \frac{N}{2} - 1$.

APPENDIX B

LEFT DOMINANCE BINARY SEARCH TREE BASED APPROACH

The maximum number of comparisons in case of left dominance binary search tree based approach is given by

$$\begin{aligned} f_{ltree} &= \left[n_{\lceil \frac{mid}{2^0} \rceil} + n_{\lceil \frac{mid}{2^1} \rceil} + \dots + n_{\lceil \frac{mid}{2^{h-1}} \rceil}\right] + [(n_1 - 1) \cdot n_2 \\ &\quad + (n_2 - 1) \cdot n_3 + \dots + (n_{K-1} - 1) \cdot n_K] \\ &= \alpha + \beta \end{aligned}$$

For maximum value of the f_{ltree} , α and β both should be maximized. The maximum value of α can be N . So we focus on maximizing β .

$$\beta = (n_1 - 1) \cdot n_2 + (n_2 - 1) \cdot n_3 + \dots + (n_{K-1} - 1) \cdot n_K$$

Now our aim is to obtain the maximum value of β so that the maximum number of dominance comparisons can be obtained. We obtain this in following manner.

A. Number of fronts is 2

The population \mathbb{P} of size N is divided in two fronts i.e. $\mathcal{F} = \{F_1, F_2\}$. Let $|F_1| = n_1$ and $|F_2| = n_2$ so $N = n_1 + n_2$.

$$\begin{aligned} \beta &= (n_1 - 1)n_2 \\ &= (n_1 - 1)(N - n_1) \\ &= n_1 N + n_1 - n_1^2 - N \end{aligned}$$

$$\frac{df_{\beta}}{dn_1} = N + 1 - 2n_1$$

$$\frac{d^2 f_{\beta}}{dn_1^2} = -2$$

The maximum value of β is achieved when $\frac{d\beta}{dn_1} = 0$ and $\frac{d^2 \beta}{dn_1^2} < 0$. Thus using this equality we get, $n_1 = \frac{N+1}{2}$ and $n_2 = \frac{N-1}{2}$. The maximum value of β is as follows:

$$\begin{aligned} f_{ltree} &= \left[\left(\frac{N+1}{2}\right) - 1\right] \left(\frac{N-1}{2}\right) \\ &= \frac{(N-1)^2}{4} \end{aligned}$$

B. Number of fronts is 3

The population \mathbb{P} of size N is divided in three fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3\}$. Let $|F_1| = n_1$, $|F_2| = n_2$ and $|F_3| = n_3$ so $N = n_1 + n_2 + n_3$.

$$\begin{aligned}\beta &= (n_1 - 1)n_2 + (n_2 - 1)n_3 \\ &= n_1n_2 + n_2n_3 + n_1 - N \\ &= (N - n_2 - n_3)n_2 + n_2n_3 + (N - n_2 - n_3) - N \\ &= Nn_2 - n_2^2 - n_2 - n_3\end{aligned}$$

The value of β will be maximized when $n_3 = 0$. Thus $N = n_1 + n_2$. From the subsection B-A, we can conclude that when the population is divided into 2 fronts then the maximum number of dominance comparison will be $\frac{(N-1)^2}{4}$. Thus for maximum number of dominance comparison, $n_1 = \frac{N+1}{2}$, $n_2 = \frac{N-1}{2}$, $n_3 = 0$.

C. Number of fronts is 4

The population \mathbb{P} of size N is divided in four fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$. $|F_1| = n_1$, $|F_2| = n_2$, $|F_3| = n_3$ and $|F_4| = n_4$ so $N = n_1 + n_2 + n_3 + n_4$.

$$\begin{aligned}\beta &= (n_1 - 1)n_2 + (n_2 - 1)n_3 + (n_3 - 1)n_4 \\ &= n_1n_2 + n_2n_3 + n_3n_4 + n_1 - N \\ &= n_2(n_1 + n_3) + n_3n_4 + n_1 - N \\ &= n_2(N - n_2 - n_4) + n_3(N - n_1 - n_2 - n_3) + n_1 - N \\ &= (n_2 + n_3)N - (n_2^2 + n_3^2) - (n_1 + n_2)n_3 + n_1 - n_2n_4 - N\end{aligned}$$

The value of β will be maximized when $n_4 = 0$. Thus $N = n_1 + n_2 + n_3$. From the subsection B-B, we can conclude that when the population is divided into 3 fronts then the maximum value of the function is $\frac{(N-1)^2}{4}$. Hence for maximum number of dominance comparison, $n_1 = \frac{N+1}{2}$, $n_2 = \frac{N-1}{2}$, $n_3 = 0$, $n_4 = 0$.

D. Number of fronts is 5

The population \mathbb{P} of size N is divided in five fronts i.e. $\mathcal{F} = \{F_1, F_2, F_3, F_4, F_5\}$. $|F_1| = n_1$, $|F_2| = n_2$, $|F_3| = n_3$, $|F_4| = n_4$ and $|F_5| = n_5$ so $N = n_1 + n_2 + n_3 + n_4 + n_5$.

$$\begin{aligned}\beta &= (n_1 - 1)n_2 + (n_2 - 1)n_3 + (n_3 - 1)n_4 + (n_4 - 1)n_5 \\ &= n_1n_2 + n_2n_3 + n_3n_4 + n_4n_5 + n_1 - N \\ &= n_1n_2 + n_2(N - n_1 - n_2 - n_4 - n_5) + (N - n_1 - n_2 - n_4 - n_5)n_4 + n_4n_5 + n_1 - N \\ &= (n_2 + n_4)N - (n_2^2 + n_4^2) - 2n_2n_4 - n_1n_4 + n_1 - N - n_2n_5\end{aligned}$$

The value of β will be maximized when $n_5 = 0$. Thus $N = n_1 + n_2 + n_3 + n_4$. From the subsection B-C, we can conclude that when the population is divided into 4 fronts then the maximum value of the function is $\frac{(N-1)^2}{4}$. Thus for maximum number of dominance comparison, $n_1 = \frac{N+1}{2}$, $n_2 = \frac{N-1}{2}$, $n_3 = 0$, $n_4 = 0$, $n_5 = 0$.

Thus we can conclude that the maximum value of β is $\frac{(N-1)^2}{4}$ which occurs when there are two fronts i.e. $\mathcal{F} =$

$\{F_1, F_2\}$. The cardinality of each front is $|F_1| = \frac{N+1}{2}$ and $|F_2| = \frac{N-1}{2}$. The maximum comparison occurs when the inserted/deleted solution dominate the solution in first front. As the tree is left dominance so in case of two fronts the inserted/deleted solution is compared with both the fronts (first F_2 then F_1) so the maximum value of $\alpha = n_1 + n_2 = N$. Thus the maximum number of comparison when dominance tree based approach is used is given by

$$f_{lree} = \alpha + \beta = N + \frac{(N-1)^2}{4} = \frac{N^2}{4} + \frac{N}{2} + \frac{1}{4}$$

APPENDIX C

RIGHT DOMINANCE BINARY SEARCH TREE BASED APPROACH

The maximum number of comparisons in case of right dominance binary search tree based approach is given by

$$\begin{aligned}f_{rtree} &= \left[n_{\lfloor \frac{mid}{2^0} \rfloor} + n_{\lfloor \frac{mid}{2^1} \rfloor} + \dots + n_{\lfloor \frac{mid}{2^h} \rfloor} \right] + [(n_1 - 1) \cdot n_2 \\ &\quad + (n_2 - 1) \cdot n_3 + \dots + (n_{K-1} - 1) \cdot n_K] \\ &= \alpha + \beta\end{aligned}$$

For maximum value of the f_{rtree} , α and β both should be maximized. The maximum value of α can be N . So we focus on maximizing β .

The maximum value of β is $\frac{(N-1)^2}{4}$ which occurs when there are two fronts i.e. $\mathcal{F} = \{F_1, F_2\}$. The cardinality of each front is $|F_1| = \frac{N+1}{2}$ and $|F_2| = \frac{N-1}{2}$. The maximum number of comparisons occurs when the inserted/deleted solution dominate the solution in first front. As the tree is right dominance so in case of two fronts the inserted/deleted solution is compared with only first front so the maximum value of $\alpha = n_1 = \frac{N+1}{2}$. Thus the maximum number of comparison when right dominance tree based approach is used is given by

$$\begin{aligned}f_{rtree} &= \alpha + \beta = \left(\frac{N+1}{2} \right) + \frac{(N-1)^2}{4} \\ &= \frac{N^2 + 3}{4} = \left\lceil \frac{N^2}{4} \right\rceil\end{aligned}$$